

3.1 BACKGROUND OF THE PROBLEM AREA

During my industrial training, my team pen-tested a lot of websites and we found out that most of these sites are vulnerable to bugs that could have been fixed by the developer during the development of the software. We decided to build a DevSecOps ci/cd (continuous integration / continuous delivery) pipeline to help integrate security practices into the DevOps process, this will help address security threats and vulnerabilities at the earliest stage of software development, ultimately improving the overall security posture of an organization.

In a traditional software development approach, security is an afterthought. Security testing is conducted late in the development process, leading to the discovery of critical vulnerabilities like SQL injection, authentication bypass, and inadequate encryption. This results in project delays, budget overruns, compliance issues, fines, and reputational damage.

In contrast, a DevSecOps approach integrates security from the beginning. The development team collaborates with security experts, employs automated security checks in continuous integration pipelines, and identifies and fixes vulnerabilities early. This approach ensures a more secure and compliant application or software with fewer security incidents, less disruption, and an improved reputation.

3.2 SYSTEM PLANNING

System description: The DevSecOps CI/CD pipeline is designed to automate and streamline the software development and delivery process while ensuring that security measures are integrated throughout. Its primary purpose is to achieve continuous integration, continuous testing, and continuous delivery of software with an emphasis on security at every stage.

Feasibility assessment: The feasibility of this project is high since the necessary tools, such as a Linux cloud server, GitHub, Jenkins, SonarQube, ZAP scan, etc., are readily accessible, with the exception of Jira, which requires a paid subscription. Additionally, I possess all the skills and knowledge required for successfully developing the project.

Methodology: In this project, the Agile software development approach was implemented since Agile methodologies are characterized by their focus on iterative development, constant feedback, and responding to change.

Requirement Analysis:

System requirement: The pipeline should integrate with tools for static code analysis to identify and mitigate security vulnerabilities in the source code, implement automated DAST to assess running applications for security issues, safely manage and store secrets, such as API keys and credentials, conduct automated security scans when there is change in source code and Implement a system for tracking, prioritizing, and addressing security vulnerabilities.

User requirement: Users should have the ability to customize and extend the pipeline to accommodate specific project needs, A user-friendly and intuitive dashboard or interface should be available for configuring and monitoring the pipeline and users should be able to create issue tickets for vulnerabilities found.

Software requirement: This comprises all the required software packages for constructing this system: GitHub, Google Cloud Platform, a Linux-based server, Visual Studio code, Jenkins, Sonarlint, Sonarqube, Trufflehog, Jira, Owasp-Zap .

3.3 SYSTEM DESIGN

System Overview: The system architecture of a DevSecOps CI/CD (Continuous Integration/Continuous Deployment) pipeline is designed to facilitate the automation of the software development and delivery process while integrating security practices at every stage. Here is an overview of the system architecture of a DevSecOps CI/CD pipeline

Components:

Source Code Repository: Central repository for source code (e.g., Git) where developers collaborate and store their code.

CI/CD Server: The CI/CD server is responsible for managing the pipeline's workflow. Popular CI/CD tools include Jenkins, Travis CI, CircleCI, and GitLab CI/CD.

Build Automation: A build server or automation tool (e.g., Jenkins) compiles, builds, and packages code. Security checks are performed during the build process to identify vulnerabilities or code quality issues.

Identifying secrets: safely manage and store secrets, such as API keys and credentials,

Automated Testing: Automated Security tests, such as static code analysis and dynamic application security testing (DAST), are an integral part of the pipeline.

Logging and Monitoring: Project management and issue-tackling tools (e.g Jira)

System Workflow:

- Developers commit code changes to the source code repository.
- The CI/CD pipeline is triggered automatically or manually when changes are detected.
- The pipeline initiates the build process, including security checks.
- Scans for secrets are triggered
- Automated testing is performed to ensure code quality, functionality, and security.
- The system logs and displays vulnerabilities
- The user creates issue tickets for significant vulnerabilities
- The developer resolves the issue

Flowchart:

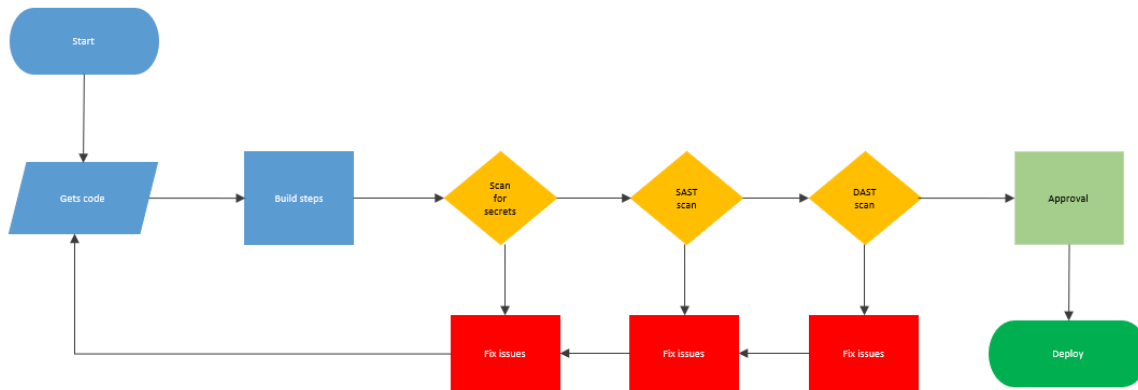


Figure 3.1 DevSecOps Flowchart

3.3 SYSTEM IMPLEMENTATION

Preparing the necessary tools:

During the setup phase, I created two Linux servers and, utilizing a headless environment (without a graphical user interface), I installed a suite of software tools, including Jenkins, Trufflehog, OWASP ZAP, and SonarQube. These installations necessitated the presence of additional prerequisites such as Java, PostgreSQL, pip, and various other tools, all of which I successfully installed. Additionally, I obtained a Git repository by forking the DVWA (Damn Vulnerable Web Application) repository, which serves as a vulnerable web application for testing purposes.

Building the project:

I created a Jenkins project and successfully integrated SonarQube with my Git repository. I also added all the required tools for the project and updated my Jenkinsfile as depicted in Figure 3.1.

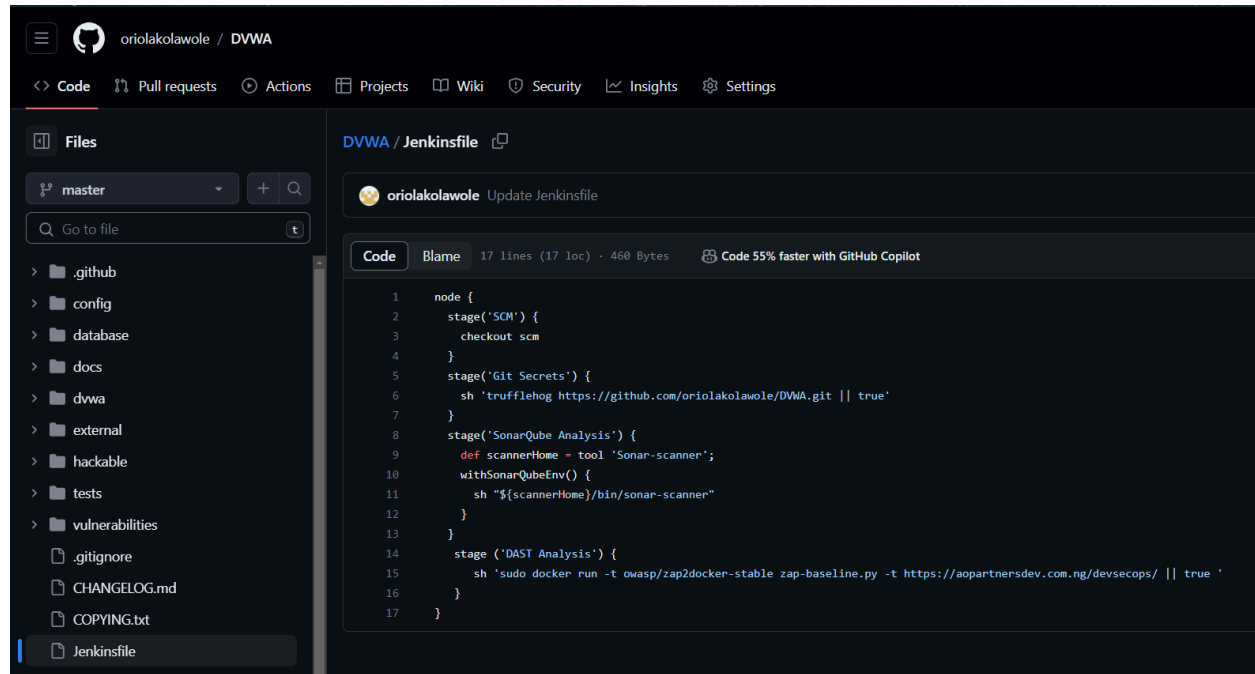


Figure 3.2 Jenkinsfile

A Jenkinsfile is a configuration file used in Jenkins, a popular continuous integration and continuous delivery (CI/CD) automation tool. It is written in a domain-specific language (DSL) called "Groovy." Jenkinsfiles are used to define the build and deployment pipeline as code, allowing developers and teams to specify how Jenkins should build, test, and deploy their software projects. The Jenkins code you provided is a simple pipeline for performing security scans on a Git repository. The pipeline has four stages:

SCM stage

The SCM stage uses the checkout scm directive to check out the Git repository to the Jenkins agent. This is the first stage in the pipeline, and it is necessary for all subsequent stages to be able to access the code.

Git Secrets stage

The Git Secrets stage uses the sh directive to run the TruffleHog tool. TruffleHog is a tool that scans Git repositories for leaked secrets, such as passwords, API keys, and SSH keys. The trufflehog <https://github.com/oriolakolawole/DVWA.git> || true command will scan the specified Git repository for leaked secrets. The || true part of the command tells Jenkins to continue with the next stage of the pipeline even if TruffleHog finds any leaked secrets.

SonarQube Analysis stage

The SonarQube Analysis stage uses the def directive to define a variable called scannerHome. This variable contains the path to the SonarQube scanner. The withSonarQubeEnv() directive is then used to set the necessary environment variables for the SonarQube scanner. Finally, the sh "\${scannerHome}/bin/sonar-scanner" command is used to run the SonarQube scanner. The SonarQube scanner will scan the code for security vulnerabilities and generate a report.

DAST Analysis stage

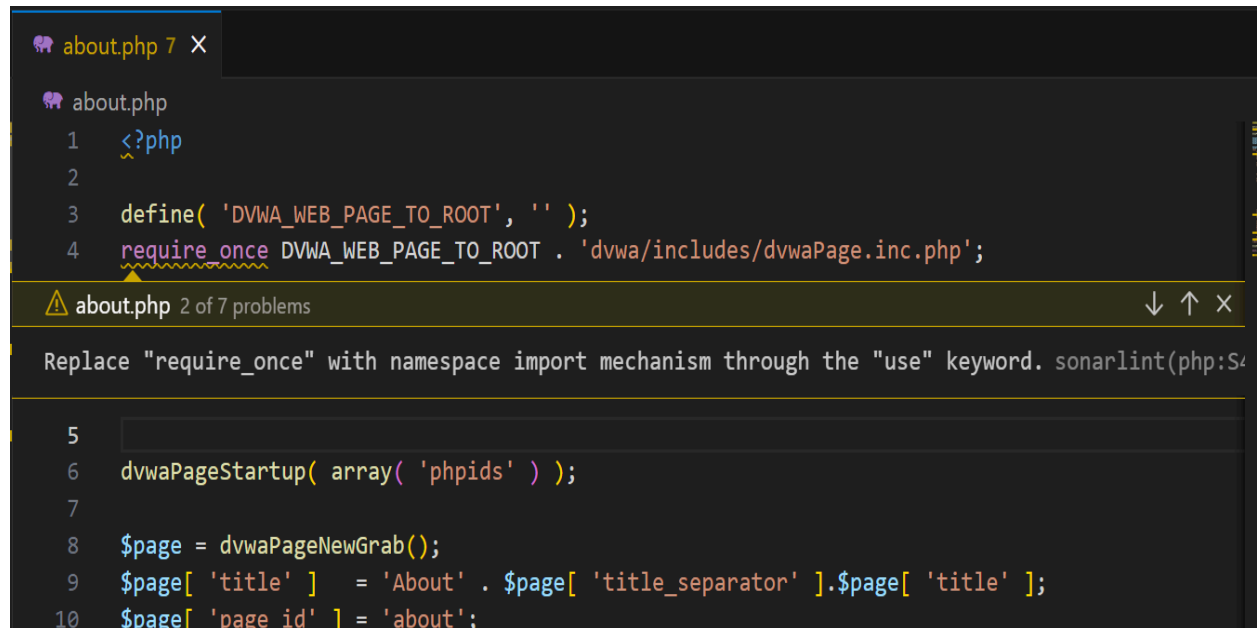
The DAST Analysis stage uses the `sudo docker run -t owasp/zap2docker-stable zap-baseline.py -t https://aopartnersdev.com.ng/devsecops / || true` command to run a DAST scan on the web application hosted at the specified URL. The OWASP ZAP tool is a popular tool for performing DAST scans. The `zap-baseline.py` script is used to create a baseline of the web application's security posture. This baseline can then be used to track the results of future DAST scans. The `|| true` part of the command tells Jenkins to continue with the next stage of the pipeline even if the DAST scan finds any vulnerabilities.

Testing:

In this stage, I conducted unit testing for each component of the pipeline individually, directly from the Linux terminal, before executing comprehensive tests for the entire system. The pipeline's flawless operation was confirmed. It is hosted on a Google Cloud server and possesses the flexibility to run on any device, ensuring compatibility across various platforms. Here are some sample results:

3.4 SAMPLE RESULT

Coding Phase: Using require and include functions may lead to file inclusion attacks



```
about.php 7 X
about.php
1  <?php
2
3  define( 'DVWA_WEB_PAGE_TO_ROOT', '' );
4  require_once DVWA_WEB_PAGE_TO_ROOT . 'dvwa/includes/dvwaPage.inc.php';
5
6  dvwaPageStartup( array( 'phpids' ) );
7
8  $page = dvwaPageNewGrab();
9  $page[ 'title' ] = 'About' . $page[ 'title_separator' ].$page[ 'title' ];
10 $page[ 'page id' ] = 'about';
```

about.php 2 of 7 problems

Replace "require_once" with namespace import mechanism through the "use" keyword. sonarlint/php:S4

Figure 3.3 Coding Phase

During this phase, a plugin or extension is added to the IDE the developer is using. This plugin or extension can help to identify and correct potential issues in the code, such as syntax errors and security vulnerabilities.

In the example above in figure 3.3 we found out that the `require_once` function can lead to file inclusion attacks.

File inclusion attacks (FIAs) are a type of web application security vulnerability that allows an attacker to inject malicious code into a web application by including arbitrary files. FIAs can be exploited by attackers to steal sensitive data, execute arbitrary code, or even gain complete control over the web server.

Require and include functions are PHP functions that allow developers to include the contents of other files into their PHP scripts. These functions can be very useful for developing complex web applications, but they can also be exploited by attackers to launch FIAs.

To exploit a FIA, an attacker would first need to identify a web application that uses the require or include functions without properly validating the user input. Once the attacker has identified a vulnerable web application, they can inject malicious code into the application by passing a carefully crafted file path to the require or include function.

For example, imagine a web application that uses the following PHP code to display a user's profile picture:

```
<?php
```

```
$filename = $_GET['filename'];
```

```
include($filename);
```

```
?>
```

If an attacker is able to control the filename variable, they can inject malicious code into the application. To solve this the developer has to ensure files are included from trusted locations and use input validation functions to ensure that the file passed to the require or include function is valid.

Git Secret: Encrypted data found

```
✓ ▾ trufflehog https://github.com/oriolakolawole/DVWA.git || true — Shell Script 9
2446 <k>descr</k><v><![CDATA[&]]></v></e></xjxobj>';
2447 - $exploits[] = "hi" said the mouse to the cat and \'showed off\' her options';
2447 - $exploits[] =
'eZtwEI9v7nI1mV4Baw502q0hmGZ6WJ0ULN1ufGmwN5j+k3L6MaI0Hv4+R1Oo42rC0KfrwUUm5zXOfy9Gka63m02fds
Sp52nhK0Jsniw2UgeedUvn0SXfNqc/z13/6mVkc7uVN63o5J8xzK4inQ1raknqYEwBHvBI8WgyJ0wKBMZQ26Nakm96
3jRb18Rzv6hz1nlf9cAOH49EMiD4vzd1g==';
2448 - $exploits[] = "European Business School (ebs)";
2449 - $exploits[] = "Deutsche Journalistenschule (DJS)";
2450 - $exploits[] = "Cambridge First Certificate FCE (2000)";
2451 - $exploits[] = 'Universität Karlsruhe (TH)';
2452 - $exploits[] = 'Psychologie, Coaching und Training, Wissenserlangung von
Führungskräften, Menschen bewegen, Direktansprache, Erfolg, Spaß, Positiv Thinking and
Feeling, Natur, Kontakte pflegen, Face to Face Contact, Sport/Fitness (Fussball,
Beachvolleyball, Schwimmen, Laufen, Krafttraining, Bewegungsübungen uvm.), Wellness &
Beauty';
2453 - $exploits[] = 'Großelternzeit - (Sachbearbeiter Lightflne)';
2454 - $exploits[] = '{HMAC-SHA1}{48de2031}{8AgxrQ==}';
2455 - $exploits[] = 'exchange of experience in (project) management and leadership •
always interested in starting up business and teams • people with a passion • new and
lost international contacts';
2456 - $exploits[] = 'Highly mobile (Project locations: Europe & Asia), You are a team
player';
```

Figure 3.4 Secrets

During this phase, if a developer makes changes to the code repository, an automated scan is triggered. This scan specifically examines the code for sensitive information such as API keys, JWT tokens, any type of token, encrypted data, and more. In the event that such sensitive data is discovered, an issue ticket is created, and efforts are made to address and resolve the identified concerns.

In the example provided, encrypted data is visible, which ideally should remain concealed. If it becomes visible, it is crucial to verify that the encryption algorithm is strong and reliable, and to ensure that the encryption key is securely stored.

SAST Scan: Potential SQL injection issue

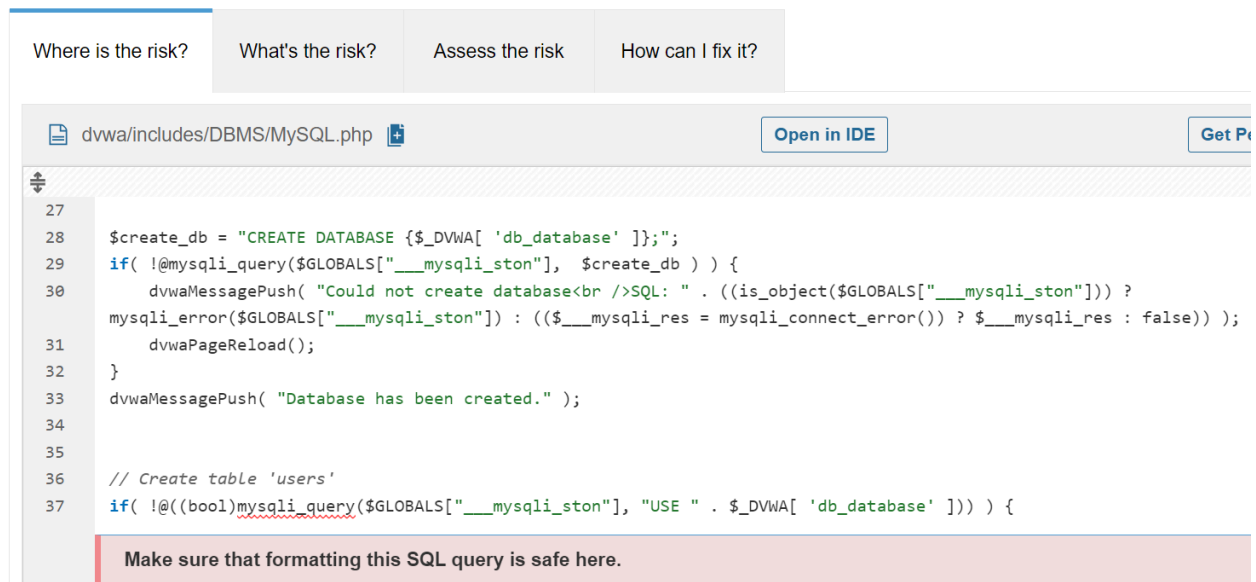


Figure 3.5 SAST scan

SAST stands for Static Application Security Testing. It is a software testing technique that analyzes source code to identify potential security vulnerabilities. SAST tools are used to scan code for a variety of vulnerabilities. SAST tools can be used to identify vulnerabilities early in the development process before attackers can exploit them. This can help to reduce the cost of fixing security issues and prevent data breaches.

In Figure 3.5, the SAST scan detected a MySQLi query and prompted us to review its formatting for safety measures. This action aims to prevent potential SQL injection vulnerabilities in the future.

DAST scan: Vulnerability scan

```
✓ sudo docker run -t owasp/zap2docker-stable zap-baseline.py -t https://aopartnersdev.com.ng/devsecops/ || true -- Shell Script
62 https://aopartnersdev.com.ng/ (200 OK)
63 https://aopartnersdev.com.ng/ (200 OK)
64 https://aopartnersdev.com.ng/ (200 OK)
65 https://aopartnersdev.com.ng/ (200 OK)
66 https://aopartnersdev.com.ng/ (200 OK)
67 WARN-NEW: Missing Anti-clickjacking Header [10020] x 3
68 https://aopartnersdev.com.ng/ (200 OK)
69 https://aopartnersdev.com.ng/devsecops/ (200 OK)
70 https://aopartnersdev.com.ng/devsecops/login.php (200 OK)
71 WARN-NEW: X-Content-Type-Options Header Missing [10021] x 6
72 https://aopartnersdev.com.ng/ (200 OK)
73 https://aopartnersdev.com.ng/devsecops/ (200 OK)
74 https://aopartnersdev.com.ng/devsecops/dvwa/css/login.css (200 OK)
75 https://aopartnersdev.com.ng/devsecops/dvwa/images/login_logo.png (200 OK)
76 https://aopartnersdev.com.ng/devsecops/dvwa/images/RandomStorm.png (200 OK)
77 WARN-NEW: Information Disclosure - Debug Error Messages [10023] x 1
78 https://aopartnersdev.com.ng/ (200 OK)
```

Figure 3.6 DAST scan

DAST stands for Dynamic Application Security Testing. It's a type of security testing that assesses an application while it's running to identify vulnerabilities by sending requests and analyzing responses. This method simulates attacks from the outside, helping to uncover security issues and weaknesses in real time.

In the example above, we're being alerted that our website lacks crucial headers such as anti-clickjacking and the X-Content-Type-Options header. Additionally, it highlighted information disclosure within debug errors

Resolving Issues using kanban methodology

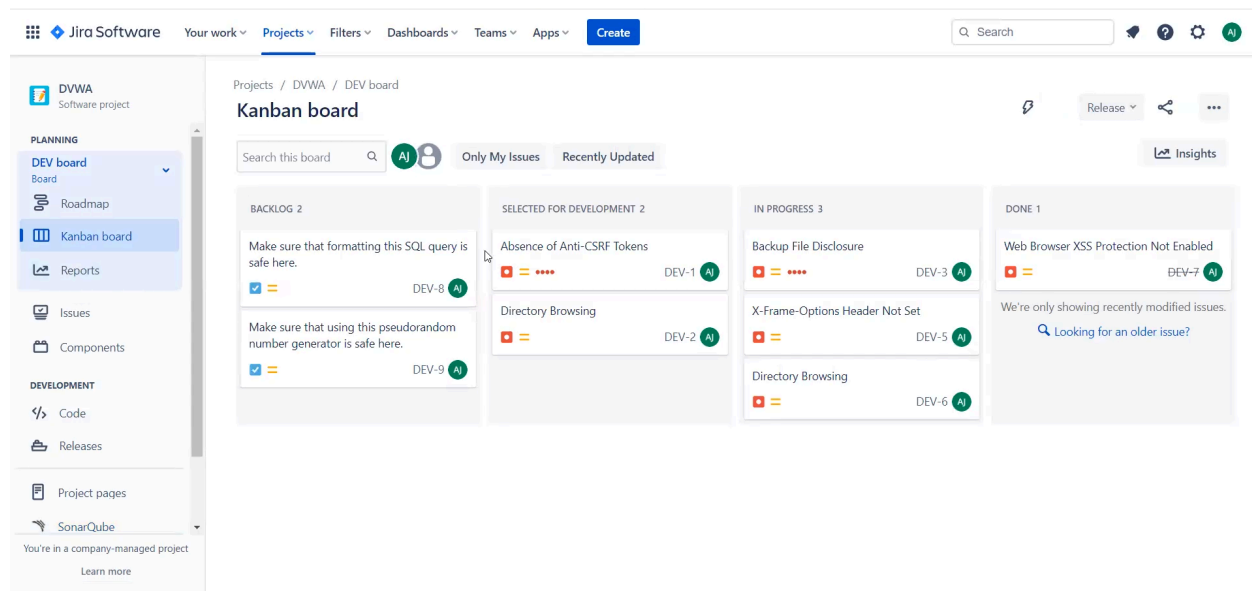


Figure 3.7 Resolving issues

Kanban is a project management methodology that visualizes work as it moves through different stages on a board. It helps teams manage tasks by visualizing the workflow, limiting work in progress, and enhancing efficiency and transparency.

In this project, we implement the Kanban methodology to address issues. We collect vulnerabilities from various scans and generate separate issue tickets for each identified concern. All these issues are showcased on the backlog. The chosen ones slated for resolution are transitioned to the subsequent phase and allocated to a developer. Subsequently, these tasks progress to the "In Progress" phase as the assigned team begins working on them. Finally, the issues move to the next phase upon completion of the tasks.